



**US Army Corps
of Engineers**
Waterways Experiment
Station

Technical Report ITL-97-6
September 1997

Migration of WAM to Scalable Computing Environments

by John E. West, Robert E. Jensen, Louis H. Turcotte

DTIC QUALITY INSPECTED 2

Approved For Public Release; Distribution Is Unlimited

19971016 060

The contents of this report are not to be used for advertising, publication, or promotional purposes. Citation of trade names does not constitute an official endorsement or approval of the use of such commercial products.

The findings of this report are not to be construed as an official Department of the Army position, unless so designated by other authorized documents.



PRINTED ON RECYCLED PAPER

Migration of WAM to Scalable Computing Environments

by John E. West, Robert E. Jensen, Louis H. Turcotte

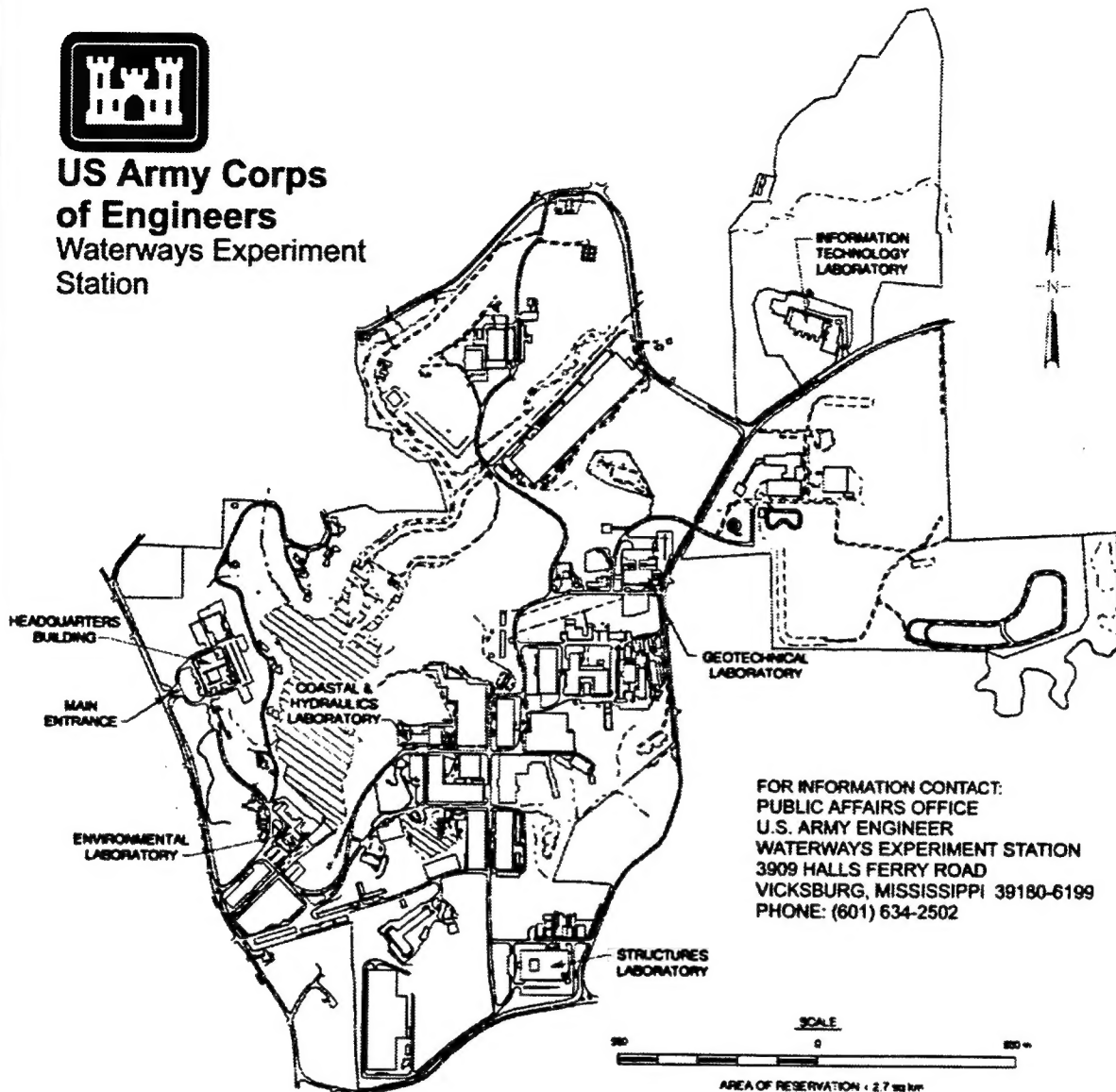
U.S. Army Corps of Engineers
Waterways Experiment Station
3909 Halls Ferry Road
Vicksburg, MS 39180-6199

Final report

Approved for public release; distribution is unlimited



**US Army Corps
of Engineers**
Waterways Experiment
Station



FOR INFORMATION CONTACT:
PUBLIC AFFAIRS OFFICE
U.S. ARMY ENGINEER
WATERWAYS EXPERIMENT STATION
3909 HALLS FERRY ROAD
VICKSBURG, MISSISSIPPI 39180-6199
PHONE: (601) 634-2502

Waterways Experiment Station Cataloging-in-Publication Data

West, John E.

Migration of WAM to scalable computing environments / by John E. West, Robert E. Jensen, Louis H. Turcotte ; prepared for U.S. Army Corps of Engineers.

37 p. : ill. ; 28 cm. — (Technical report ; ITL-97-6)

Includes bibliographic references.

1. Ocean waves — Forecasting. 2. Water waves — Forecasting. I. Jensen, Robert E. II. Turcotte, Louis H. III. United States. Army. Corps of Engineers. IV. U.S. Army Engineer Waterways Experiment Station. V. Information Technology Laboratory (U.S. Army Engineer Waterways Experiment Station) VI. Title. VII. Series: Technical report (U.S. Army Engineer Waterways Experiment Station) ; ITL-97-6.

TA7 W34 no.ITL-97-6

Contents

Preface	vii
Summary	viii
1 Introduction	1
1.1 The WAve Model (WAM)	1
1.2 Common High Performance Computing Software Support Initiative (CHSSI)	4
1.3 Goals	4
2 Approach	6
2.1 WAM: Baseline Software Configuration and Performance	6
2.2 Modernization Implementation Plan	9
2.3 Software Considerations	10
2.4 Previous Efforts	10
3 Phase I: Parallelization of the Job Stream	12
3.1 Pipeline Parallelism	12
3.2 Complications	13
3.3 Results	15
4 Phase II: Domain Decomposition	16
4.1 Domain Decomposition	16
4.2 Complications	19
4.3 Pilot Implementation Results	20
4.4 Completion of Phase II	22
4.4.1 I/O	22
4.4.2 Memory	23
4.4.3 Task Control	23
4.4.4 Optimization of Communications	23
4.4.5 Verify and Validate	24
5 Time and Resources	25
5.1 Computational Resources	25
5.2 Human Resources	26

6 Beyond Phase II	28
Bibliography	29
SF-298	30

List of Figures

1.1	Three level grid nest for coastal wave height forecast in the East China Sea.	2
2.1	Three level grid nest for wave height forecast for the Del Marva Peninsula during hurricane Luis.	7
2.2	Simultaneous forecast of multiple regions in the Atlantic.	11
3.1	Comparison of original and Phase I execution; messages between processors in (2) shown by arrows.	13
3.2	Graph of runtime tasks for WAM on 3 processors.	14
4.1	Block structure of the Atlantic basin grid with four blocks.	17
4.2	Comparison of Phase I and Phase II execution; messages between processors shown by arrows.	18
4.3	Block structure of the Atlantic basin grid with four blocks distributed to four processors.	18
4.4	Wall clock time on SGI Origin and PCA systems.	22

List of Tables

2.1	Schematic of hurricane Luis serial execution.	7
2.2	Performance of the four most active WAM routines (CRAY C90).	8
2.3	Wall clock time for the hurricane LUIS simulation (CRAY C90).	8
2.4	Schematic of hurricane Luis parallel execution.	9
4.1	WAM performance on an R10K Origin system (wall clock times).	21
4.2	WAM performance on an R8K PCA system (wall clock times).	21
5.1	CPU minutes consumed during development.	25

Preface

This report¹ presents results and analysis from the modernization of the Wave Model (WAM) wave height prediction code for parallel computation as part of the DoD Common High Performance Computing Software Support Initiative (CHSSI).

This research was performed at the U.S. Army Engineer Waterways Experiment Station (WES), Vicksburg, MS, by the following personnel: John E. West, DoD High Performance Computing (HPC) Major Shared Resource Center (MSRC), Information Technology Laboratory (ITL), WES; Dr. Robert E. Jensen, Coastal and Hydraulics Laboratory (CHL), WES; and Dr. Louis H. Turcotte, ITL, WES. This work was funded in part by the DoD High Performance Computing Modernization Office under the auspices of the CHSSI program, and was made possible by a grant of computer resources at the DoD CEWES MSRC. Dr. Joseph W. McCaffrey, Jr., Naval Research Laboratory, Stennis Space Center, MS, was the Computational Technology Leader for Climate/Weather/Ocean Modeling and Simulation (CWO). The work was under the direction of Dr. N. Radhakrishnan, Director, ITL.

During preparation of this report, Dr. Robert W. Whalin was Director of WES. COL Bruce K. Howard, EN, was Commander.

¹The contents of this report are not to be used for advertising, publications, or promotional purposes. Citation of trade names does not constitute an official endorsement or approval of the use of such commercial products.

Summary

Recent reviews in Department of Defense (DoD) strategic interests place additional emphasis on near-shore, or littoral, operations. This has created a significant Army interest in coastal wave predictions for sustainment of operations and has significantly increased the Navy's needs for improved coastal wave predictions. Both services use a European-developed code called the WAVE Model, or WAM, for oceanic wave prediction. Although initially developed for deep ocean, global forecasts, research indicates that extension of WAM to handle near-shore forecasts may be very successful, if adequate computational power is available.

This report documents results of work currently underway as part of the DoD's scalable software initiative to improve the performance of WAM by a factor of two to four times current operational performance. Techniques to achieve this level of performance on the emerging breed of scalable, commodity-processor based parallel computing systems in a way which is both efficient and portable are detailed and placed in the context of an overall design philosophy. Development in two phases is discussed. Phase I focuses on the parallelization of the simulation job stream so that each of the successively finer grid nests used in a forecast can be computed simultaneously. Phase II development extends this parallelism by performing computations for each grid on multiple processors. Preliminary results are presented which demonstrate excellent performance improvements in initial evaluations, and specific recommendations for further enhancements are provided.

Chapter 1

Introduction

1.1 The WAVE Model (WAM)

Recent reviews in Department of Defense (DoD) strategic interests place additional emphasis on coastal, or littoral, operations. This shift has created a significant Army interest in coastal wave predictions for sustainment of operations, and has significantly increased the Navy's needs for improved coastal wave predictions. Both services use a European generated computer code called the WAVE Model, or WAM [6][9], for oceanic wave prediction. This code was initially developed for deep ocean, global forecasts. Active R&D efforts are underway within the Navy and Army to extend and improve this class of model for littoral operations and planning. Initial investigations suggest this may be effective, if adequate computational power is available [11].

Wind-wave models such as WAM involve physical processes which operate over a large range of time and space scales. Wave characteristics at a specific time and place are often the result of winds and waves originally generated days ago and thousands of kilometers away. Yet wave propagation remains dependent upon local effects due to the shape of the ocean bottom and, close to the shore, can require local grid resolutions as fine as 100m. WAM must therefore compute solutions over spatial resolutions ranging from 100m to (typically) 60 km, and involve integration of the field equations over times ranging from hours to weeks. Wave height prediction on these widely varying temporal and spatial scales is accomplished by performing simulations in several stages. Each stage uses a grid which is finer than the previous grid and covers an area successively closer to the area in which the final prediction is desired. For example, Figure 1.1 shows a three level grid nest for the prediction of wave heights near the shore in an area of the East China Sea. The coarsest grid level encompasses the entire Pacific Ocean, while the next two successively finer grids focus in on regions closer to the shore. These simulations are said to be "nested" because each coarse grid completely contains all finer grids for which it produces boundary conditions. Nesting the simulations in this way enables waves generated in an area remote from the area in which a prediction is desired to influence conditions near the shore; it also minimizes computational time by calculating the most accurate solutions only near the regions of operational interest. A simulation may contain more (or less) grid levels than shown in Figure 1.1 depending upon the requirements of a particular forecast.

Although developed for deep ocean, global forecasts, recent investigations indicate that

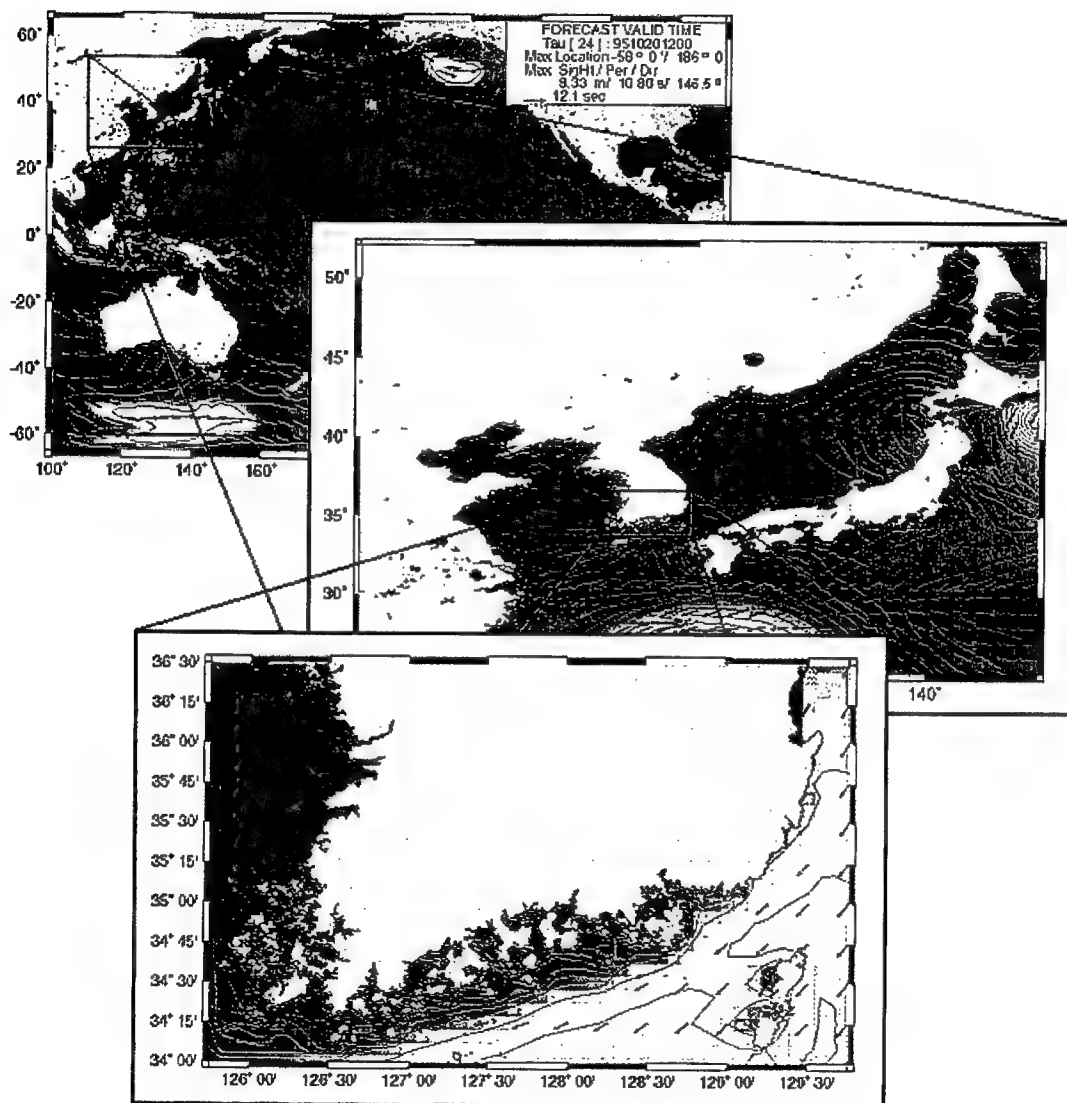


Figure 1.1: Three level grid nest for coastal wave height forecast in the East China Sea.

this model can be extended into much shallower water for coastal simulations [11]. These studies also indicate that adequate treatment of the coastal wave prediction problem will require significant increases in computational power. First, simulation of wave activity in coastal regions will require local grid meshes an order of magnitude finer than presently used in order to define depth gradients, shoreline configuration, and current fields. Moreover, increased spatial resolution may be needed to resolve near-shore wave dynamics in a coupled modeling system, further increasing the computational burden. Finally, a significant increase in model accuracy, desired for increased confidence in operational forecasts, will require an improved representation of the collision integral estimated to consume about 45% of the computational demands in the application.

However, simply adding additional code to satisfy the increased computational requirements may not yield an application model which is useful in operational logistics forecasts. These advancements must produce forecasts within operational time constraints. Forecasts provided by WAM are of vital importance to fleet operations, and must be delivered in a timely fashion. WAM is used by Fleet Numerical Meteorological and Oceanographic Center (FNMOC) and the Naval Oceanographic Office (NAVOCEANO) to perform continuous forecasts of wave heights in areas of military significance (*i.e.*, areas of responsibility, or AORs). Timely transmittal of results to the field is of the highest priority for these simulations, and several tradeoffs have been made to reduce the wall clock time. For example, some of the most critical physical processes in the model have been implemented parametrically because of the high computational cost of more accurate representations. A reduction in wall clock time will enable use of higher-order representation of these features, increasing the accuracy of computed results and enhancing the dependability of operational predictions. Furthermore, more and more regions of the world become candidates for near-shore wave predictions as the political climate continues to evolve, while computational resources have remained fairly fixed. Reduction of the wall clock time required for forecasts in the various AORs will allow additional regions to be simulated during a single forecast cycle. For example, a global model with six regional simulations and two ultra-high resolution forecast scenarios run twice daily by NAVOCEANO requires approximately 3.5 hours of computation per watch cycle. A reduction of only 20% in the run time will enable another region to be simulated in the same amount of time.

A significant factor in this project is, therefore, the reduction of wall clock time for each of the simulations while retaining a suitable level of forecast accuracy. Standard rules of thumb reveal that a 48 hour simulation, from global to the near-shore grid, must be completed in no more than 100 minutes (wall clock time) with an accuracy of $\pm 0.5\text{m}$ at coarse resolutions and $\pm 0.25\text{m}$ desired in the littoral zone. During each stage of WAM development the source code was optimized for single vector processor environments such as the CRAY C90. Researchers at WES have made further improvements by reducing the wall clock time as much as 20% (mostly through environment-specific modifications such as the use of high-speed I/O routines). The result is an application which makes effective use of the vector environment for which it has been tailored, but further refinements are not likely to deliver the level of performance needed to support the desired technology enhancements. The current work, following trends in the High Performance Computing (HPC) industry, focuses upon migrating WAM from a single processor vector environment to a scalable computing environment.

1.2 Common High Performance Computing Software Support Initiative (CHSSI)

The migration of WAM to a scalable computing environment is partially funded by the Common High Performance Computing Software Support Initiative (CHSSI), a component of the DoD HPC Modernization Program (HPCMP). Recognition by the DoD of long-emerging trends in the HPC industry toward use of mass-market, off-the-shelf components in high performance computations led to the creation of four Major Shared Resource Centers (MSRCs), operated by the three services (two MSRCs in the Army, one each in the Air Force and Navy). A significant portion of the computational resources available in each of these four centers is composed of scalable, RISC-processor based machines like the Silicon Graphics, Inc., Power Challenge Array (PCA), the CRAY T3E, and the IBM SP [8]. These machines represent a transition the primary production supercomputing platform in the DoD, the vector architectures typified by the CRAY Y-MP and C90. The DoD recognized that, with this shift in computational resources, a significant amount of software re-engineering, or "modernization", would be required to take advantage of the new architectures and ensure mission success. The CHSSI program will produce a set of principal DoD applications programs that run efficiently on scalable HPC systems. As summarized in HPCMP documentation [4]:

"The software initiative has been instituted by the HPC Modernization Office to help support the overall program vision of developing scalable software applications for DoD, building user expertise in the DoD laboratories and centers, and applying HPC computation and communications to maintain technological superiority of war fighting systems. The goals for CHSSI are: (1) to develop critical DoD software applications that can execute in a scalable computing environment (from the "desktop" to the largest HPC architectures); (2) to foster reuse of software tools and application software components as well as appropriate use of communication standards, interface standards, and graphics visualization standards across DoD; (3) to support local HPC requirements at selected sites where there is potential for advancing DoD applications; (4) to promote the development of new software tools and application area specific software; and (5) to leverage HPC expertise and assets located in industry, academia and other federal laboratories in addition to DoD facilities."

1.3 Goals

The modernization of WAM for scalable computation will yield a faster application for modeling directional wave spectra in coarse (global) through ultra-fine scale (littoral zone) modeling of the near-shore zone. Current projections of the improvement needed to accomplish very fine scale near-shore modeling for operational forecasts involve a speedup of two to four times present computational rates. Operational forecasts made using WAM to date have been performed in a primarily single-processor vector environment. Although this environment has been utilized efficiently, very little potential for further improvements on this architecture is expected and competition for these resources throughout the DoD continues to rise. As the large scalable computational resources of the four DoD MSRCs have become available, modifying WAM to utilize these new scalable platforms has become an attractive option in terms of both resource availability and the potential for increased per-

formance through parallelism. Because of the rapidly changing nature of the computational resources deployed throughout the DoD, it is necessary that whatever modifications are made to support parallelism be portable to a variety of current and foreseeable HPC architectures, including the new generations of shared memory, distributed memory, and hybrid architectures. Also, it is desirable that enhancements to reduce execution time for an entire simulation (*i.e.*, a collection of grid nests, not just a single level) optimize the time to solution in the highest resolution domains in order to ensure swift response to logistics-over-the-shore operational planning forecast requests.

Chapter 2

Approach

Before detailing the methods used in the current work to parallelize WAM, it is important to understand how the code is currently used, and what some of its vital computational characteristics are. WAM is presently used in both forecast and hindcast prediction. The test problem used during development of this project is a simulation of hurricane Luis in the Atlantic which predicts wave heights for a region along the coast of the Del Marva Peninsula. We will use this problem as the motivating example for understanding both the original code and the changes detailed in this chapter. The domain of each simulation is shown in Figure 2.1.

2.1 WAM: Baseline Software Configuration and Performance

The hurricane Luis simulation involves three levels of grid nesting: one for the North Atlantic Ocean (referred to as the basin grid), one for the entire Atlantic coastline (the region grid), and a final nest to resolve wave heights in a near-shore subregion of the Del Marva Peninsula (the subregion grid). There are several steps which must be taken at each level of the simulation. First, two preprocessing programs are run for each level of the grid. These programs create the grid, specify the physical domain in which the simulation takes place, and initialize parameters specifying the characteristics of the solution at that level (location of wind field files, whether a given level is a shallow or deep water run, etc.). After this preprocessing step is complete, a solution is generated for the coarsest grid – in this example, the North Atlantic basin. After this solution is complete, an interpolation program is used which creates boundary conditions from the basin grid output needed at each time step of the solution for the next finest grid, the region. This process is repeated at each of the successively finer resolutions until the final solution (wave heights in the grid nearest the shore) is obtained. Table 2.1 illustrates this process for hurricane Luis, our representative example.

As mentioned above, the existing application is targeted for execution in a single-processor vector environment. Specifically, WAM has historically been run on CRAY Y-MP and CRAY C90 vector machines. Analysis on the C90 indicates that four routines account for about 82% of the application's total execution time. Table 2.2 shows accumulated times and performance in Mflops for each of these four routines. The numbers in the table indicate that WAM is a very efficient code, achieving a sustained 60% of peak processing rate in the

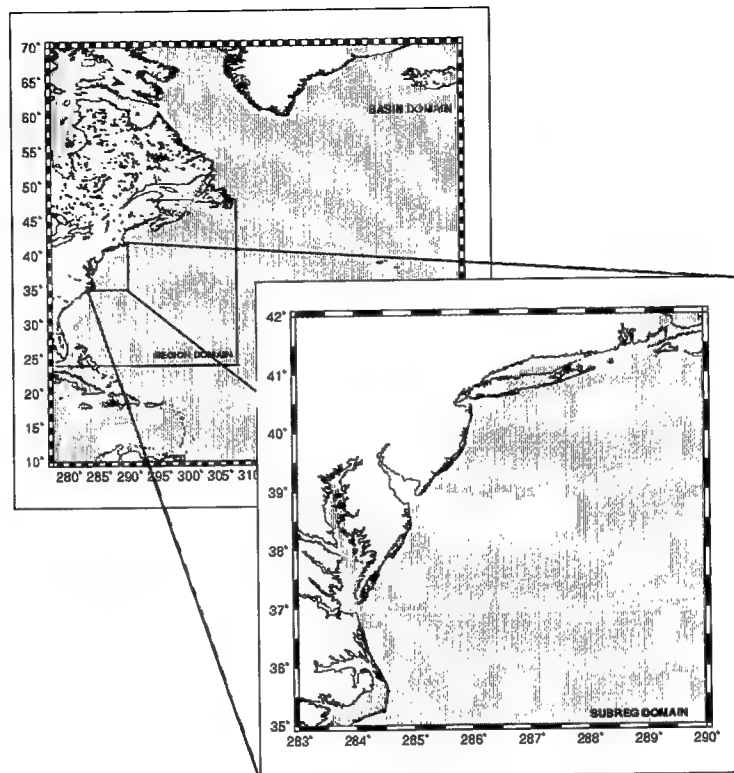


Figure 2.1: Three level grid nest for wave height forecast for the Del Marva Peninsula during hurricane Luis.

Basin					
Region					
Subregion					
	Preprocess	WAM	Interpolate	WAM	Interpolate
				WAM	Interpolate

Time



Table 2.1: Schematic of hurricane Luis serial execution.

Routine	% of Wall Time	Mflops [‡]
SNONLIN	45	600
SINPUT	23	300
IMPLSCH	9	350
PROPAGS	5	520

[‡] Peak performance per CRAY C90 processor is 958 Mflops.

Table 2.2: Performance of the four most active WAM routines (CRAY C90).

Grid Level	Wall Time (s)
Basin	830
Region	2267
Subregion	2200
Total	5297

Table 2.3: Wall clock time for the hurricane LUIS simulation (CRAY C90).

most time consuming routine. The application performs a significant amount of file I/O, which is aided on the C90 by use of the Solid State Device (SSD).

Table 2.3 shows the wall clock times (averaged over three runs) for each grid level of a 24-hour simulation running on the C90. These times do not include the preprocessing time and the time to interpolate the solutions between grid levels. These times are several orders of magnitude smaller than the WAM times, and are thus negligible. The wall clock time to complete an entire 24-hour simulation from basin to subregion is approximately 5300 seconds. The same simulation run without use of the Solid State Device (SSD) consumes approximately 15,000 seconds, a threefold increase in execution time. Note that these simulations are out-of-core solutions and are thus I/O intensive (WAM's out-of-core method will be discussed in more detail later). Also, these simulations were run interactively on the C90 in full production mode (over 99.8% sustained utilization). Although we were given special priority to ensure that the job rarely swapped out, competition for I/O resources was intense, and no doubt contributed dramatically to the huge increase in execution time for the non-SSD run.

An important observation to make from Table 2.3 is that the computation is inherently unbalanced. The basin simulation is solely a deep water calculation on a coarse grid, and takes less than half as much time as each of the remaining grid nests. This leads to a situation in which, if these three jobs are performed in parallel with no overhead, the best speedup attainable on three processors, $\frac{S_1}{S_3}$, is approximately 2.5. This will become an important factor guiding the decomposition of each problem domain in the second phase of development.

Basin		WAM (Interpolate)
Region	Preprocess	WAM (Interpolate)
Subregion		WAM (Interpolate)


Time


Table 2.4: Schematic of hurricane Luis parallel execution.

2.2 Modernization Implementation Plan

The opportunities for parallelism in this code occur at two levels. First, parallelism can be introduced by overlapping the computation of each grid level, as shown schematically in Table 2.4. The output of each grid level is a set of boundary conditions, one set for each time step in the simulation, which is used for the computation of the solution at the next level. In this case an opportunity for coarse-grained parallelism can be identified in creating a type of ocean computation pipeline. The boundary conditions at the basin scale for the first time step are created and sent to the region so that it may begin its computation immediately, rather than waiting for the entire basin computation to complete for all time steps. The basin then computes a solution for the next time step while the region computes a solution for its first time step. Once the region has generated the solution needed by the next level, boundary conditions are sent to the subregion which then begins its first computation. This process is then repeated until the computation at all levels is complete. This type of parallelism is referred to as pipeline parallelism [10]. In this case the performance benefits are derived at the task level, and it is clear that the communication patterns of this type of parallel implementation will be characterized by relatively infrequent (once per time step) messages sent (in one direction) between grid levels.

The second opportunity for parallelism lies in dividing the work of each single grid over many processors in a more fine-grained approach¹. Depending upon the nature of the computations being performed, this type of parallelism can be challenging to implement. WAM is, however, particularly well-suited to this type of optimization. The solution method is explicit, so that none of the difficulties inherent in the solution of implicit systems on parallel machines are encountered. Furthermore, because WAM was originally written for small storage computer systems, there is a built-in out-of-core solution method which splits the entire computation domain into K blocks of adjacent latitudes, each having L elements. Because the solution method is explicit, the solution for a given block at time step t is dependent only upon the solution at time $t - 1$, plus any local source terms (all source terms in WAM are local). The solution of block K is completely independent of the solution at other blocks, with the exception of one overlapping latitude on the north and south faces of each block. A solution containing K blocks can thus be effectively solved on K processors. This approach will involve larger amounts of data being communicated than the first, pipeline, approach, but each of the rounds of communication will still occur with relative infrequency at only once per time step. Also, the characteristics of the communications are good in that they

¹However, note that this is not “fine-grained” parallelism in the traditional, loop-level, sense.

only involve messages between pairs of processors to exchange adjacent latitude information or gather boundary conditions to send to the next level.

Given these two excellent opportunities for performance enhancement, our approach is centered on two primary phases of development. Phase I concentrates on the implementation of the pipeline parallelism between grid levels, while Phase II seeks to parallelize the solution computations at each level.

2.3 Software Considerations

The two key guidelines which have influenced both the design and implementation of the current modifications to WAM are portability and maintainability. We stated above that one of the goals for a modified WAM is portability to a variety of existing and future HPC machines. This has been achieved, as much as possible, using the Message Passing Interface (MPI) [3] to facilitate the exchange of data between processors used in a simulation. MPI has the advantage of presenting a common, standards-defined, interface to message passing semantics on all machines which support an MPI implementation (almost every current HPC architecture). In many cases machine vendors have implemented the functions of MPI tuned to their architectures. Thus, a call to `MPI_Send()` on a distributed memory machine may be implemented using the native message passing library on an IBM SP and using shared memory constructs on an SGI Origin 2000. MPI removes from the programmer the burden of having to create and maintain different versions of an application for different architectures and from having to acquire specialized knowledge about the nuances of data transfer on different machines.

The notion of preserving portability has further implications for code modifications. WAM was initially highly optimized for vector architectures, and performed at 60-70% of peak on a single processor of a CRAY C90. The modified WAM is to be supported on a variety of scalable, typically RISC-based, modern HPC architectures. Performance optimizations on RISC-based systems can be extremely complicated, and are sensitive to parameters which vary not only from chip to chip, but between configurations of machines with the same chip (such as cache and main memory sizes). For this reason no RISC-specific algorithmic modifications have been planned during initial development. This decision was also influenced by our recognition that this code is in use by a variety of organizations around the world, each of which would be required to maintain the code over its years of use. WAM is implemented entirely in Fortran 77, and the original authors were clearly very concerned with the maintainability of this code as it is extremely well-documented and very modular. We therefore have striven to not disturb the existing code wherever possible, and where changes have been made to concentrate on simplicity. The end result will hopefully be a code which is as easily maintained as the original.

2.4 Previous Efforts

We are aware of one other effort to parallelize WAM. This effort, undertaken by the European Centre for Medium Range Weather Forecasts (ECMWF), has concentrated on a distributed memory implementation for the Fujitsu [1]. This implementation uses a specialized message

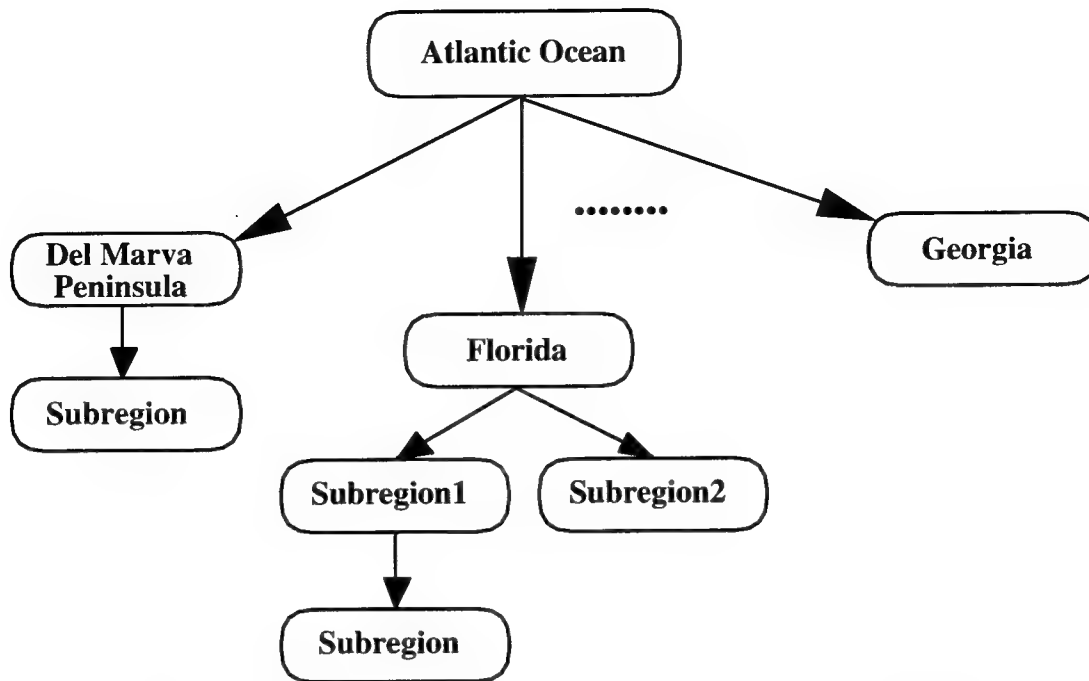


Figure 2.2: Simultaneous forecast of multiple regions in the Atlantic.

passing library created at ECMWF and has undertaken only the parallel computation of a single grid level; successively finer solutions have to be run in turn, one after the other. Furthermore, it appears that the ECMWF has not chosen to exploit the block element structure of the code in decomposing the solution domain, choosing instead to decompose the domain into groups of computation points using a separately developed algorithm.

The advantage of our approach to parallelizing both the entire job stream and individual grid solutions is that it is possible to run forecasts for multiple grid nests in different geographic regions simultaneously. For instance, in our hurricane Luis simulation a subsection of the Atlantic Ocean results are extracted and sent as boundary conditions to the computations off the Del Marva Peninsula. It is a simple matter to extract boundary conditions for other areas of the Atlantic region, say the coast of Florida, and transmit that data to another set of processors performing forecasts in this area, as shown in Figure 2.2. Likewise, more than one subregion of the Florida coastline could be computed from boundary conditions created by a single region simulation. In this way a set of geographic regions which share common areas can be computed simultaneously, greatly improving the turnaround time for the entire collection of forecasts, not just individual jobs. Furthermore, our choice of a general message passing library imparts a degree of portability not immediately available to the ECMWF effort.

Chapter 3

Phase I: Parallelization of the Job Stream

3.1 Pipeline Parallelism

Phase I of this project focused on the development of a fully parallel job stream which couples the grid nesting from low resolution areas into fine-scale resolutions. The original application computes solutions for successively finer grid levels one at a time using the interpolated solution of a coarse grid to generate boundary conditions for the next finer grid (see Table 2.1). In Phase I, solutions on each of the grid nests (three in the hurricane Luis simulation) are computed “simultaneously,” as shown in Figure 3.1 and the boundary conditions for each time step are sent between processors as they are computed via messages using MPI. (In fact, the solutions are not precisely simultaneous, as each grid level needs boundary conditions which are generated by the previous (coarser) grid. Instead, each grid level is offset one time step from the previous levels, in much the same way a modern microprocessor pipelines instructions for faster execution.) The result is a coarse grained, pipeline parallel code. Although each level receives boundary conditions at every time step of the previous level, the computation to communication ratio remains favorable. This is because both the temporal and spatial resolution of the preceding levels are coarser than the finer levels, which means that after the subregion receives boundary conditions for the first two time steps of the region it computes solutions at several intervening time steps, interpolating between the supplied sets of boundary conditions as needed. The boundary conditions are sent asynchronously from “parent,” the coarser grid, to “child,” the finer grid, which improves the communication profile of the application. The relative infrequency of messages between levels make a pipeline implementation of the WAM code effective on a range of parallel architectures, including workstation clusters.

There is a significant load balancing issue which arises in Phase I due to the difference in solution times of the original application. Table 2.3 shows that the basin scale solution requires less than half the time of the remaining regions to complete. If each task is dedicated to a single processor, boundary conditions will be sent from the basin to the region twice as fast as they can be received, potentially hobbling the message passing system buffers as pending boundary conditions pile up. Furthermore, basin computations are so much faster than the region computations, at each time step the boundary conditions will have

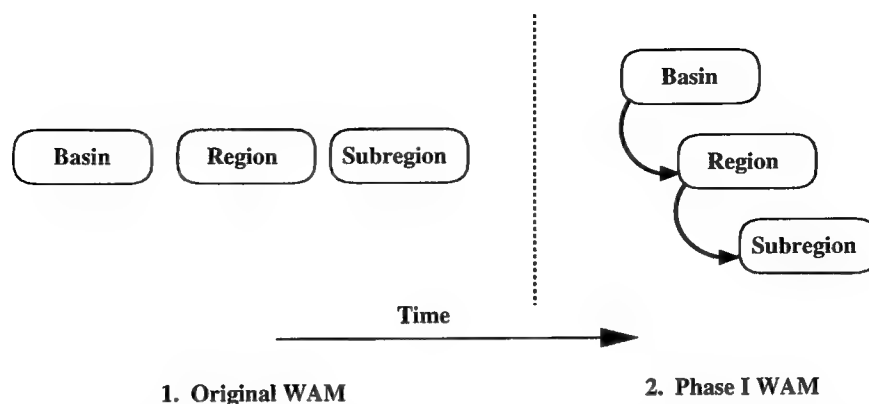


Figure 3.1: Comparison of original and Phase I execution; messages between processors in (2) shown by arrows.

to be stored in temporary buffers before posting a send to avoid overwriting data before the communication system has transferred it to the receiving processor. If the basin and the region computations get significantly out of step, this could dramatically increase the memory required to run the application. These issues were addressed by requiring that a processor wait until the previous asynchronous send is initiated before sending the boundary conditions for the next time step. For the basin this results in about 50% idle processor time, as shown in Figure 3.2, a snapshot from SGI's MPIView tool for runtime analysis. There is almost no idle time for the region and subregion solutions as the computation rates for these areas are similar. The 50% idle rate leads to a significant processor inefficiency, but was not addressed at this phase of the project because the result of Phase II will be an application with multiple processors assigned to single grid levels. It will then be possible to assign fewer processors to the solution of the basin boundary conditions and create a more balanced computation (alternatively, in the event that the basin computation required more time than the region or subregion computations, more processors could be devoted to the basin in order to balance the computation).

An additional modification was needed to complete the Phase I implementation. The original execution model called for a post-processing operation to be performed on the output of each coarse solution, temporally interpolating the boundary conditions for use by the finer grid. The use of a separate program to perform this step is not viable in a multi-processor environment as it interrupts the data flow. To avoid this problem the temporal interpolation program was incorporated into the main application as a subroutine, streamlining the execution process and facilitating multi-processor execution.

3.2 Complications

There were several small obstacles which arose in the conversion of WAM to a multi-process model. Initially WAM took input from standard input (the keyboard), and provided output on standard output (to the screen) as well as to files. Some of these files were named and explicitly opened, while others were implicitly opened. Because all of the grid levels start at the same time in the parallel version, the input data for each grid level problem was moved

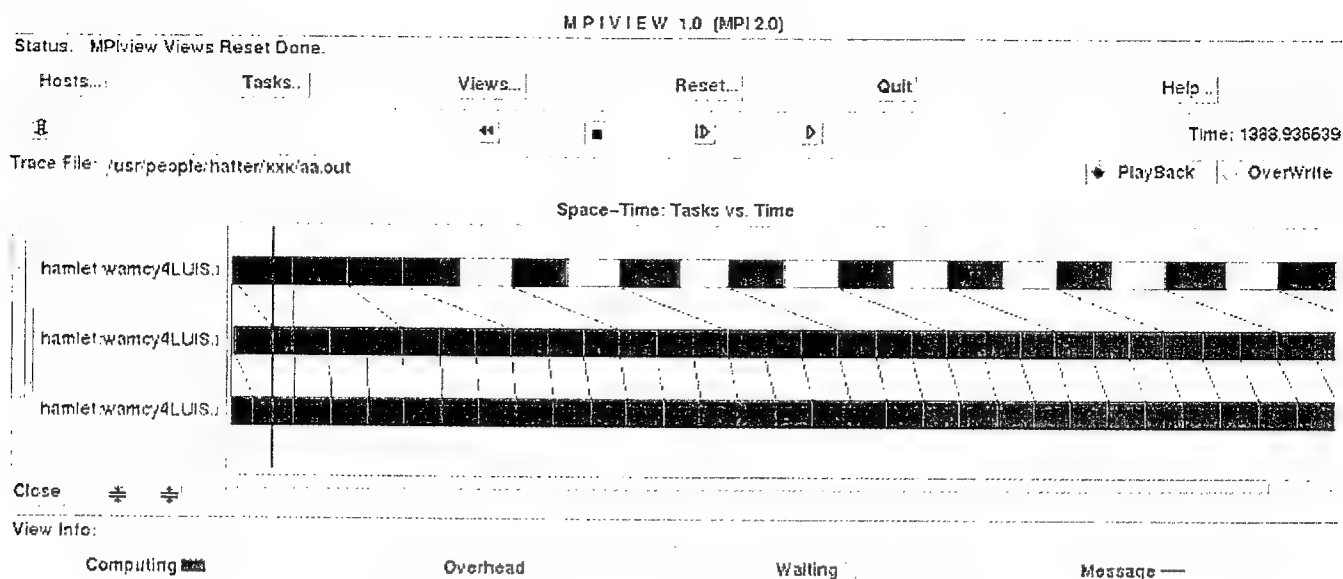


Figure 3.2: Graph of runtime tasks for WAM on 3 processors.

to separate files which were read by each processor at the beginning of execution. Output file units which shared the same integer identifier across grid levels were simply incremented by an amount dependent upon the processor number. This shifted the file identifiers such that none of the processors tried to do I/O on the same units. Although this is adequate for a temporary patch, this solution has several long term problems. First, it doesn't scale well, in that there is a practical limit on the file identifier numbers. Secondly, not all processors on parallel machines have full I/O capability. These issues are resolved in Phase II with a more robust I/O solution.

All of the initial Phase I development was done on the CRAY C90 using MPI. After the parallel version ran and provided correct answers, development was moved to an architecture more representative of the project's target execution platform, an SGI Power Challenge Array (PCA). For the most part the transition was straightforward, but there was one problem related to the internal time-keeping mechanism. WAM computes a variety of date and time step variables and stores them as integers. For example, dates are stored in integers which contain the year, month, day, hour, and minute; 9508290100 represents a solution time of one o'clock on the morning of August 29, 1995. This poses no problem on the C90, which uses a 64-bit integer representation. The PCA, however, uses a 32-bit integer by default, which is inadequate to represent the date. Moreover, simply promoting all integers to 64-bit caused the program to produce incorrect answers. The difficulty was finally resolved by changing all date quantities to `INTEGER*8`.

Finally, in combining the three different simulations into one executable most of the arrays in the application had to be re-dimensioned and new variables introduced. The original application was run with a setup script which automatically changed the dimensions of solution arrays to match the size of the problem being solved for that grid nest and re-compiled. For example, the variables NBLO and NIBLO contain the number of blocks and the number of points per block, respectively. Both of these quantities vary from level to level.

Each of these variables was replaced in the parallel application with two new variables. The first, MAX_NBLO and MAX_NIBLO store the maximum values over all grid levels, and are used for dimensioning arrays. LOC_NBLO and LOC_NIBLO store the values to be used by each grid nest. This substitution was repeated for all of the problem variables which are used to define arrays. Note that this does result in some wasted memory on at least two of the grid levels (whichever two do not require the maximum dimension), but this cannot be avoided without transitioning to a language which supports dynamic memory allocation (*i.e.*, FORTRAN 90).

3.3 Results

As mentioned earlier, the initial development was done on the CRAY C90 to avoid dealing with complications from the architectural differences between the PCA and the CRAY. To give an idea of the performance differences, a test run was done on the CRAY before shifting to the PCA. The test run was done as an in-core solution (meaning the number of blocks at each grid level was 1) using the SSD and three processors, one for each grid level. This run was approximately 3 times faster, at 1645 seconds, than the out-of-core single processor job using the SSD. Earlier we asserted the maximum speedup on three processors is limited to 2.5 due to differences in the execution times of the three grid resolutions. We speculate that the additional speedup in this case is due to the sharp reduction in I/O which results from not storing intermediate data for blocks of the out-of-core solution to the SSD.

After porting to the PCA, test runs were made and results validated on the new architecture. The PCA used is comprised of two chassis, each having sixteen 90 MHz R8000 processors with a 4 MByte secondary cache, 16 KByte instruction and data caches, and 8192 MBytes of main memory. The execution time on three processors using SGI's MPI 2.0 is 15,900 seconds, or 3.18 times the single stream C90 run. Individual processors on the PCA are approximately 3 times slower (peak) than those of the C90, and this version did not take advantage of any of the asynchronous I/O capabilities of SGI's IRIX operating system (as was the case for the C90). Given these differences, the Phase I implementation was considered generally successful, although it was very clear that additional enhancements would be necessary to meet the development goals in Section 1.3.

Chapter 4

Phase II: Domain Decomposition

Implementation of Phase II modifications was slightly more tedious than in Phase I. Whereas the approach in Phase I is the concurrent solution of each grid level in a multi-level simulation, the approach in Phase II is to accelerate the solution of each grid level by dividing the work among multiple processors. As we will see, the software infrastructure to exploit this type of parallelism was already present in the application as part of the out-of-core solution capability. All that is required is to facilitate the passage of data from the latitudes which are shared by neighboring blocks via messages rather than the file mechanism used in the original code.

The type of parallelism being exploited in Phase II is applicable to cluster-type configurations, but will most likely perform best on systems with moderately-well-coupled processors. The characteristics of Phase I and Phase II together indicate a range of architectures on which the final parallel system will be effective. The best match of communication patterns to architecture is SMP cluster machines typified by the Silicon Graphics, Inc. (SGI) Power Challenge Array (PCA). However, the system can naturally benefit from a uniformly high bandwidth between processors as found on a range of machines including the IBM SP, the CRAY T3E, and the SGI Origin.

4.1 Domain Decomposition

Phase II development focuses on decomposing each grid level and solving it on more tightly coupled parallel processors. Depending upon the nature of the computations being performed, this type of parallelism can be extremely difficult to implement (see, for example, [5]). However, WAM is particularly well-suited to this type of optimization. The solution technique in the application is explicit, so that none of the difficulties inherent in the solution of implicit matrix systems on parallel machines are encountered. Furthermore, WAM was originally designed with the flexibility to solve large ocean problems on older, smaller memory vector machines such as the CRAY X-MP and thus supports out-of-core solutions. Out-of-core solutions are produced by dividing the solution domain into blocks of roughly equivalent numbers of neighboring sea points along groups of lines of latitude, shown in Figure 4.1. WAM then computes solutions for a block at a time, rotating through all blocks until an entire time step has been solved. The solution of each block is written to a file before moving to the next block so that the necessary information will be available to generate the

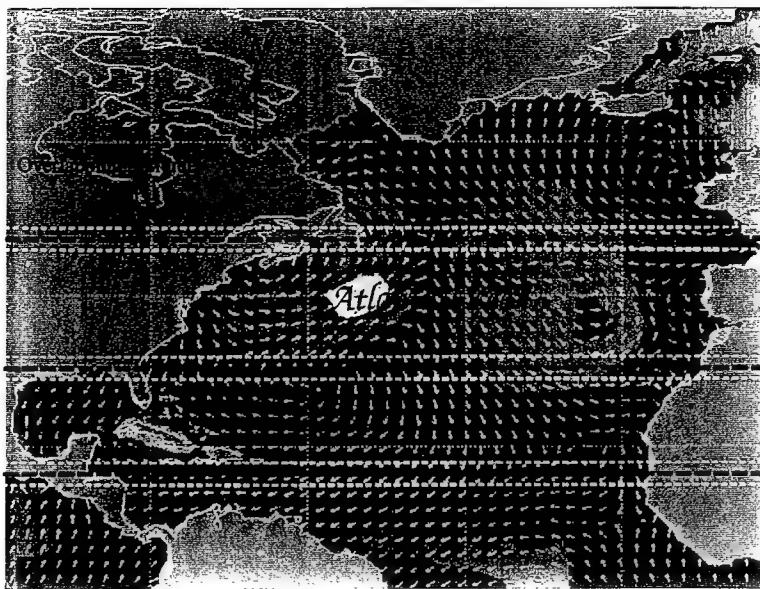


Figure 4.1: Block structure of the Atlantic basin grid with four blocks.

solution for that block at the next time step. In order to facilitate advection of wave energy between neighboring blocks of the domain, special arrays are set up to store data from the last line of latitude of the previous block and the first latitude of the next block for use in the solution of the current block (see Figure 4.1). This decomposition, referred to in WAM documentation as the "block element structure," can be directly exploited in computing a single solution on multiple processors. For each grid level, the solution domain is blocked as if an out-of-core solution is to be performed, with the number of blocks equal to the number of processors desired for the solution. Each block is then assigned to a processor, and the computations for all blocks are performed in parallel. The new computation model is shown in Figure 4.2. Communications between processors are denoted by arrows, which also indicate the direction of data flow as show in Figure 4.3 for the Atlantic Ocean. Note that it is possible to relax the requirement that the number of processors equal the number of blocks in the solution at the expense of more complex control logic. This has not been done in the current implementation because blocking is performed in the preprocessing step which must be run before every simulation, and it is likely that the user will know in advance the number of processors available for the simulation. Nevertheless, relaxation of this requirement may prove to be desirable in an operational forecast environment, and future efforts should consider development in this direction.

The domain decomposition approach to parallelism involves communication of more data than the pipeline approach, but each round of communication still occurs with relative infrequency at only once per time step. The characteristics of the communications are also good in that they involve messages between pairs of processors to exchange adjacent latitude information or to assemble boundary conditions for transmission to a finer grid level (some global communication does take place during job initialization as the master processor distributes various parameters to the workers, but these are one-time only communications).

The primary task of Phase II was to shift from using intermediate files and temporary

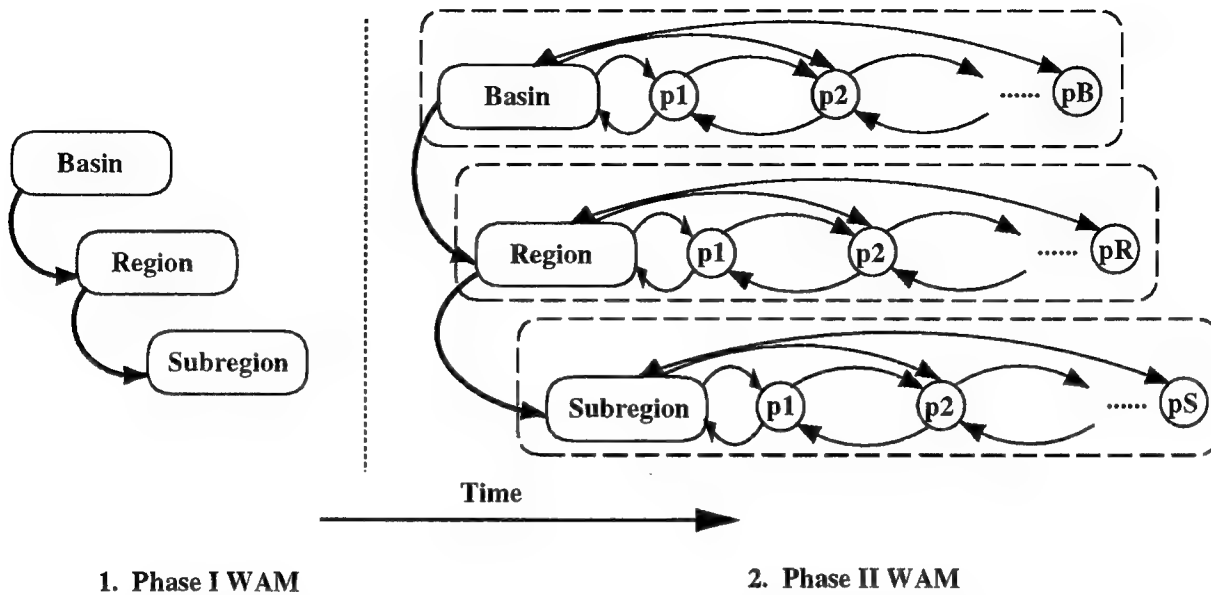


Figure 4.2: Comparison of Phase I and Phase II execution; messages between processors shown by arrows.

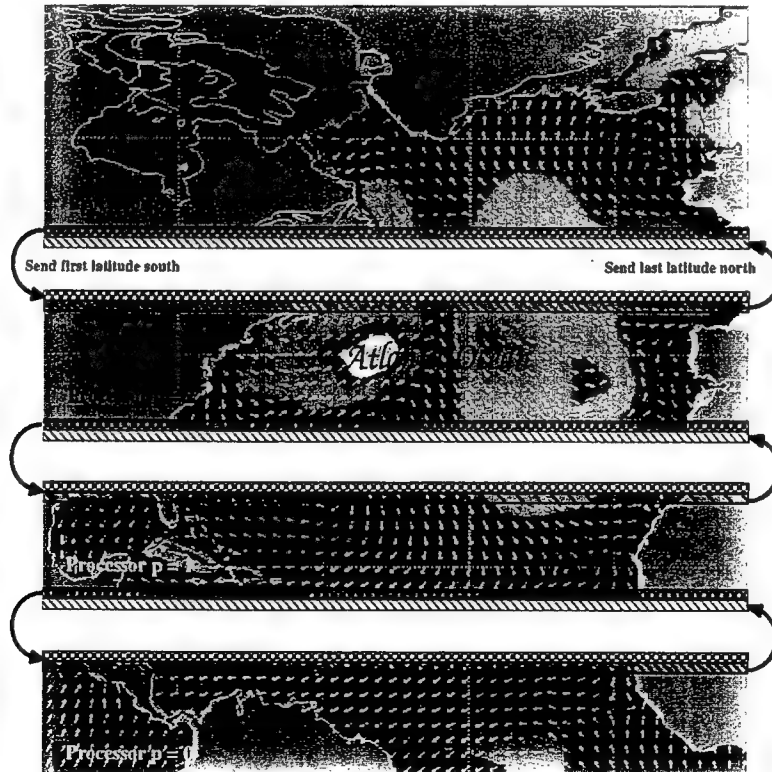


Figure 4.3: Block structure of the Atlantic basin grid with four blocks distributed to four processors.

arrays for the storage of overlapping latitudes (used in the north-south energy advection computation) to exchanging these data via messages. This was generally straightforward, although care had to be taken to ensure that the information was passed at the correct times to prevent introducing a phase shift into the solution. Each grid solution is parallelized using a master-worker model, where each of the processors dedicated to the solution of a single grid are coordinated by a master processor (which also does work on the solution). This master processor is responsible for distributing initialization information from the input files and new wind fields at each time step, and collects the boundary conditions from each processor for transmission to the master processor of the next finer grid level.

4.2 Complications

There were a range of minor issues which arose in the conversion of WAM to use a finer-grained parallel solution model. Whereas a Phase I simulation could potentially use only a limited number of processors, the Phase II implementation can use many more processors in solving each grid level. This promises to create I/O problems on several fronts. First, the final implementation of WAM must be portable, and there are a range of parallel machines which do not support I/O to the disk subsystem by all processors. Furthermore, the Phase I approach of simply shifting the integer file identification units to prevent duplication of file identifiers is not general, and will become problematic for large numbers of processors. These issues were addressed by replacing all implicit file open operations with explicit operations which open the same file units for each processor, but now each file is contained in a unique directory for every processor. Thus, temporary variables written out to unit 20 on processor 1 are written to 1/fort.20, 2/fort.20 for processor 2, and so on. Where possible the application has been further modified to eliminate file output altogether. Much of the file I/O performed by this application has to do with the out-of-core solution method, which has been replaced in favor of solving all blocks simultaneously. Finally, where possible input of problem data such as start up values and wind fields is handled by that particular region's master processor.

Another issue which had to be addressed was communication of boundary conditions between groups of processors dedicated to different geographic regions. Given the relative infrequency of communications it was decided that all such communications would take place only between the master processors of each region. After computation of the boundary conditions for a given time step is completed by all processors in a grid level, the master processor for that level assembles the partial sets of boundary conditions from the responsible processors and sends them *en masse* to the master processor of the next finer grid level. This solution is simpler to implement than alternatives (such as having each processor send partial boundary conditions to other regions individually and assembled at the receiving end), but forcing all communications between regions through a single processor may create a significant bottleneck in large simulations of the form of Figure 2.2. Further testing is needed to investigate this phenomenon.

4.3 Pilot Implementation Results

It should be noted that the current Phase II implementation is essentially a pilot implementation. While WAM is now fully parallelized and results of the new code are correct for the test case, a significant amount of development remains. The majority of these tasks can be classified as generalizing WAM beyond the specific three nest test case used in development. A detailed look at the tasks remaining in the full implementation of Phase II is provided in the next section. First, it will be useful to examine performance results for this stage of Phase II.

In the interim between Phase I and Phase II development the primary application platform, the SGI PCA, became an extremely busy machine and competition for resources dramatically slowed the development process. Phase II development efforts therefore shifted to the newer (and less used) SGI Origin 2000 system at the CEWES MSRC. This system is comprised of two chassis each having sixteen 195 MHz R1000 processors with a 4 MByte secondary cache, 16 KByte instruction and data caches, and 8192 MBytes of main memory. The performance of this machine on WAM is significantly lower than that of the R8000, by a factor of almost two. The three processor PCA solution which took on 4.5 hours takes approximately 9 hours on three processors of the Origin. It should be noted that because this machine was to be used solely as a development platform no effort was put into choosing the optimal set of compiler directives; we simply used the same directives as on the PCA. Previous experience with this code has shown that as much as a five-fold performance improvement (observed on the PCA) can be gained with the right compiler directives, so the potential for additional improvement on this architecture exists. Despite this caveat, it is nonetheless instructive to consider some performance figures from test runs on the Origin.

As SGI's implementation of MPI was not available during development for the Origin, all message passing is done using the 64-bit version of the generic version of MPI, MPICH [2], compiled with the shared memory device driver (the Origin is a distributed shared memory machine). The solution domain was blocked for various numbers of processors using the preprocessing programs as shown in Table 4.1 (recall that the number of processors used in a given grid level is equal to the number of blocks). The table provides the wall clock time in seconds for a series of runs of the same 24-hour simulation of hurricane Luis on the Origin. The table also gives an indication of the speedup for the job as a whole by dividing the quotient of the six-processor execution time, T_6 , and the n -processor execution time, T_n , by the ratio of the number of processors to six,

$$P_r = \frac{\frac{T_6}{T_n}}{n/6}. \quad (4.1)$$

P_r is thus a "pseudo-efficiency" measure, giving an indication of how well the solution time scales with the number of processors.

Table 4.1 illustrates that the effectiveness of dedicating additional processors to the solution of a given grid nest varies according to where those processors are added. As an example, consider the thirteen-processor case in which the number of processors dedicated to the region and subregion (roughly) doubles from the six-processor case. Recall from Table 2.3 that the basin takes approximately half as long as the remaining grids to solve, so the performance of the six-processor case in Table 4.1 is limited by the speed of the region

Nprocs	Time (s)	P_r †	Number of blocks		
			Basin	Region	Subregion
6	14340	1.00	2	2	2
8	10380	1.03	2	3	3
13	5400	1.23	4	5	4
15	4785	1.22	3	6	6
21	2760	1.50	4	9	8
25	2700	1.38	8	9	8

† See Equation 4.1.

Table 4.1: WAM performance on an R10K Origin system (wall clock times).

Nprocs	Time (s)	Number of blocks		
		Basin	Region	Subregion
8	6540	2	2	2
15	3780	3	6	6

Table 4.2: WAM performance on an R8K PCA system (wall clock times).

and subregion. Doubling the number of processors devoted to these grids therefore halves (approximately) the time to solution. This solution is now fairly well balanced, and no significant advantage is gained by adding more processors to the basin. This is illustrated by the last line in Table 4.1. The extra four processors in the basin solution (over those employed in the 21-processor case) are not effectively utilized in terms of the performance of the overall simulation, a fact illustrated by the decrease in P_r for this case as the time to solution remains constant.

Application test runs were also performed on the PCA for the Phase II version of the code for comparison to Origin results. These results, provided in Table 4.2, were obtained using a beta release of SGI's MPI 3.0 on one chassis of the array. Figure 4.4 illustrates these wall clock times for both the Origin and PCA systems.

Finally, note that the 21-processor case reports a solution time of 2760 seconds, roughly half the single processor C90 time for the same simulation using the SSD. We have achieved both our performance goal of a $2\times$ performance improvement over the original code and have a 48-hour solution time of 90 minutes (less than the goal of 100 minutes). It is important to recall, furthermore, that these results have been achieved with no architecture-dependent optimizations (*e.g.*, caches, etc.). Moreover, several enhancements, detailed below, remain to be implemented. Overall these initial results are encouraging and indicate that WAM will run very effectively in production computations on RISC-based multi-processor systems.

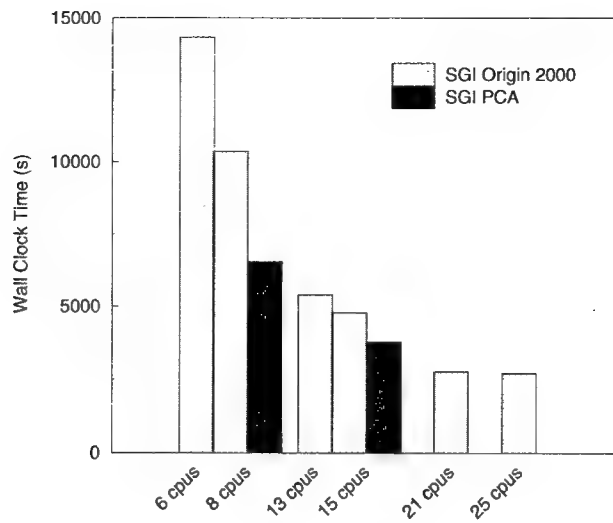


Figure 4.4: Wall clock time on SGI Origin and PCA systems.

4.4 Completion of Phase II

The pilot implementation of Phase II was concluded due to time constraints to accommodate a shift in personnel. Many modifications are either not implemented or not complete. Some of these, like generalization of task control, are necessary for WAM to function conveniently in a production forecast environment while others, like memory and I/O optimizations, will have a direct impact on the performance of the application.

4.4.1 I/O

As discussed, the solution of individual grids is parallelized in WAM using a master-worker model in which all communication between grid levels is performed by the region master processors. Also discussed above was the notion that, because not all parallel machines support symmetric access to the I/O subsystem by all processors, the master processors should be responsible for all I/O in the application. This idea has been only partially implemented. For example, while the master processors do currently take responsibility for reading the various restart files at the beginning of a simulation period and for distributing the wind fields at each time step, each processor still reads its own copy of the input file (input which used to come from standard input). The master processors do currently assemble data for writing to a single output file (unit 22), while several other files, particularly the restart files for the next simulation period, are still written in pieces by each processor. A complete Phase II application will concentrate I/O access to ensure that all file operations are carried out through the designated master processors for each region. This is not expected to add significantly to application run time as most modifications will only introduce communication steps at the beginning, during setup and initialization, or at the end, when data must be assembled from each processor to produce the various output and restart files needed for the next simulation period. Specifically, the "MAP," "SWE," "SPE," and "SWS" files created

on units 20, 21, 25, and 26, and the restart files written to unit 15, must be handled correctly [7]. Also, the routine `OUTINT()` must be modified to produce a single complete output file.

Remnants of scratch files used by the out-of-core solution method still exist and can be removed, eliminating more unnecessary I/O. Where I/O must be performed in the code every effort should be made to take advantage of asynchronous libraries on the target platforms. Unfortunately asynchronous I/O routines typically vary from machine to machine, and so special efforts will have to be made (probably in the form of conditionally compiled code via a `#ifdef` mechanism) to ensure that the completed application remains portable. WAM currently supports some asynchronous I/O for the CRAY C90.

4.4.2 Memory

There are many two- and three-dimensional floating point arrays in the application which are still dimensioned as `IXLG(MAX_NIBLO,MAX_NBLO)` on each processor. In this case the array `IXLG` is declared to be a two-dimensional array with *number of points per block* rows and *number of blocks* columns. This construct is a remnant of the out-of-core solution method, and is no longer necessary as each processor contains only one block. These arrays can now be dimensioned as, for example, `IXLG(MAX_NIBLO)`. This change will result in significant memory savings and potentially enhance the cache behavior of the application. The modifications, though numerous and tedious to make, are not difficult and can be automated.

4.4.3 Task Control

The routine `PARWAMCHIEF()` controls the assignment of regions and blocks within a region to groups of processors, as well as the appointment of region masters and creation of the various communicators and MPI variables which will be used during a simulation. In the present implementation it is necessary to set the majority of these variables explicitly. The code thus requires modification and recompiling every time the number of blocks or processors changes. (Note that some modifications for new simulations will always need to be made to WAM manually if the problem sizes or blocking changes; all of these variables are found in `jwparall.h`. This process, though somewhat cumbersome, is part of the original WAM methodology and an accepted step in the execution process.) Generalization of the controlling routine to set up these task allocation and control variables automatically will be straightforward. This is essentially a cosmetic step, but one which is necessary if the code is to be used conveniently in a production forecast environment.

4.4.4 Optimization of Communications

In the transition from Phase I to Phase II several new communications, both collective and point-to-point, were added to the application. Where possible, these communications were overlapped with computations if such optimizations could be made without significant development time. Now that the pilot implementation is complete these communications need to be re-examined to determine whether more of them can be made asynchronous, and to what advantage. This step must be undertaken with care, however, as it is possible to introduce temporal phase shifts in the solution.

4.4.5 Verify and Validate

Finally, before Phase II can be confidently recommended as complete, a preliminary verification and validation of model results against the original version of WAM must be conducted. During development the code was tested in hindcast mode, comparing results for the first 24-hour period of the hurricane Luis simulation with results from the original application. The entire ten-day Luis simulation needs to be conducted in its entirety to ensure that the application passes information between simulation periods properly and to verify that the numerical characteristics of the solution are not changed under severe weather conditions. Following this, a suitable number of different simulations should be done and compared with results from the accepted version of WAM.

Chapter 5

Time and Resources

As discussed in the introduction, the modernization of WAM for scalable computing environments is funded by the HPCMP's CHSSI program. One of the stated goals of CHSSI is the development of user expertise within the various research agencies of the DoD in porting and developing codes for commodity processor-based scalable computing environment. An important part of that expertise is the ability to accurately estimate the level of computational and human resources required to achieve a desired level of performance in a scalable computing environment. For that reason it is instructive to spend time discussing the resources required in the development of WAM to its current state (naturally the Phase II numbers will have to be revised to reflect the resources required to carry that phase to completion).

5.1 Computational Resources

Table 5.1 summarizes the processor time consumed on all three machines utilized during development to date. The total estimated HPC computation time used during development to date on all platforms is 38,315 minutes (or roughly 27 days of dedicated computation).

Phase I development involved two computational platforms: the CRAY C90 and the SGI Power Challenge Array, both at the CEWES MSRC. The C90 was used for the initial implementation of Phase I, rather than moving directly to the PCA, to avoid the additional complications which are typically encountered in migrating between such disparate architectures (word size, file formats, etc.). The development team was given interactive privileges on the C90 which enabled this machine to be effectively used for multi-processing (this

Resource	CPU Time (min)
CRAY C90	9644
SGI PCA	13297
SGI Origin	15374
Total	38315

Table 5.1: CPU minutes consumed during development.

development pre-dated the shift in C90 allocation policies which created a more favorable environment for parallel processing by all users). A Cray version of MPI was not available on this machine at the time of development; the generic version of MPI, MPICH, was used instead and compiled with the p4 communication device driver. During the initial part of Phase I, 9644 CPU minutes were consumed; this usage was spread over four months, with the heaviest processor use concentrated in a single month (7320 minutes in July of 1996). Note that this time is the sum of the processor time accumulated on, for the most part, each of three CPUs (so that, for example, a three processor job which takes ten minutes consumes a total of thirty processor minutes).

Following the completion of the initial stage of Phase I, development efforts were moved to the SGI PCA. This architecture is representative of the scalable parallel processing platform for which the new version of WAM is targeted. A total of 13,297 minutes on three CPUs was accumulated during the completion of Phase I modifications on this machine. Much of these resources were consumed in tracing and repairing incorrect code behavior caused by the integer size differences (discussed in Section 3.2), with the remainder used during various simulations to test the accuracy of results from the modified code.

The migration of development efforts to the PCA occurred immediately following the installation of this machine at the CEWES MSRC, during a period when the machine was available to pioneer users. Shortly after the PCA was released for general use its resources were totally consumed (the job load on each sixteen-processor chassis currently averages more than 25 jobs), with a resulting severe degradation of the development environment. Fortunately, Phase II development was started about the same time that a new 32-processor SGI Origin 2000 system was installed at the CEWES MSRC and made available for pioneer use. Phase II development was moved to the Origin system to take advantage of the extremely high processor availability. The development advantage derived from the infrequent use of the Origin by other users during the pioneer period more than outweighed the performance disadvantage incurred in moving from the PCA to this system (recall that observed WAM run times on the Origin, without making any "Origin-specific" optimizations, are roughly twice those for the PCA on the same number of processors). The bulk of current Phase II modifications were made in approximately five weeks, during which time 15,374 minutes, or roughly 11 days, of CPU time (three to 25 processors) was used.

5.2 Human Resources

The bulk of WAM development has been concentrated in two periods of effort, one for each of Phase I and Phase II (pilot implementation). Implementation of Phase I took place over a sixteen week period, during which approximately twelve weeks were devoted exclusively to this project. Some of the sixteen week period was devoted to exploring the application and gaining an understanding of its computational and algorithmic characteristics as well as planning the course of development. WAM is composed of 55 routines (roughly 11,000 lines of declarations and code), of which seven were substantially modified in Phase I. We will not quote statistics such as "lines of code changed," because this type of metric gives little indication of the level of effort expended to make those changes, and can make a software project appear unduly complicated. In fact, a lines changed measure for this project would indicate a startling number of changes, but most of these arise from "bookkeeping"

alterations like removing repeated common blocks in individual routines to include files.

Roughly six weeks have been devoted to the pilot implementation of Phase II, of which four were spent in active code development (with the balance spent in planning). In Phase II five routines were modified substantially from the Phase I implementation, including two routines which were not changed in Phase I.

Chapter 6

Beyond Phase II

Once Phase II development has been completed, the application will enter its third major period of development. The CHSSI program specifies that each project must conclude with a production version of the application available for users. Phase III development must therefore include interaction with members of the Climate/Weather Oceans (CWO) modeling community with an operational or R&D interest in WAM to determine their specific needs and how the initially released parallel version of WAM might be enhanced to better serve those needs. This step is essentially a “productization” of the application, and might involve an alpha release to selected users for initial design and performance feedback. Following reception of this input and subsequent modifications, a beta release of the code to interested parties in the CWO community might be conducted to fully test the parallel code in operational and research environments on a variety of architectures. At some point in this process, either pre-alpha or pre-beta release, the entire code must undergo a rigorous verification and validation to both demonstrate the effectiveness of the parallel solution approach and to build confidence in the modified application. Phase III might also conceivably involve publication of results in both the high performance computing and ocean modeling literature as well as at community (such as the Supercomputing series of conferences) and DoD events.

Bibliography

- [1] J. Bidlot, B. Hansen, and P. Janssen. Modifications to the ECMWF WAM code. Technical Report No. 232, Research Department, European Centre for Medium-Range Weather Forecasts, Reading England, February 1997.
- [2] N. Doss, W. Gropp, E. Lusk, and A. Skjellum. A model implementation of MPI. Technical Report MCS-P393-1193, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439, 1993.
- [3] Message Passing Interface Forum. MPI: A message-passing interface standard. Technical Report CS-94-230, Computer Science Department, University of Tennessee, Knoxville, TN, May 5 1994. Appears in the International Journal of Supercomputing Applications, Volume 8, Number 3/4, 1994. The standard is available at <ftp://netlib2.cs.utk.edu/mpi/mpi-report.ps>.
- [4] R. S. Foster. Department of Defense scalable software initiative. In *DoD High Performance Computing Modernization Office Web Site*, November 1996. URL <http://www.hpcm.dren.net/Htdocs/CHSSI/chssi-intro.html>. Accessed 16 APR 1996.
- [5] G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon, and D. Walker. *Solving Problems on Concurrent Processors*. Prentice Hall, 1988.
- [6] The WAMDI Group. The WAM model – a third-generation ocean wave prediction model. *Journal of Physical Oceanography*, 18:1775–1810, 1988.
- [7] H. Gunther, S. Hasselmann, and P. Janssen. Wamodel cycle 4. Technical Report No. 4, Deutsches KlimaRechenZentrum, Hamburg. 91 pp., 1991.
- [8] HPCMO. HPC hardware listing. In *DoD High Performance Computing Modernization Office Web Site*, March 1997. URL <http://www.hpcm.dren.net/Htdocs/hardware.html>. Accessed 21 APR 1996.
- [9] G.J. Komen, L. Cavaleri, M. Donelan, K. Hasselmann, S. Hasselmann, and P. Janssen. *Dynamics and Modelling of Ocean Waves*. Cambridge University Press, 1994.
- [10] C. M. Pancake. Is parallelism for you? *IEEE Computational Science and Engineering*, 3(2):18–37, 1996.
- [11] M. Y. Su, Y. L. Hsu, C. L. Vincent, and R. E. Jensen. Developing a joint Army/Navy coastal wave prediction program – a planning report. Technical Report NRL/MR/7330-95-7686, Naval Research Laboratory, Stennis Space Center, MS, 1996. 103pp.

Destroy this report when no longer needed. Do not return it to the originator.